*Volume 31 (2012), Number 7 pp. 1–13* COMPUTER GRAPHICS *forum*

# Appearance Stylization of Manhattan World Buildings

C. Li and P.J. Willis and M. Brown

University of Bath
United Kingdom

**Figure 1:** *We use a novel representation for Manhattan World building stylization. The base model (first picture) is decorated using characteristic appearances captured from different example buildings. Note the result models differ in decoration details.*

**Abstract**

*We propose a method that generates stylized building models from examples. Our method only requires minimal user input to capture the appearance of a Manhattan world building, and can automatically retarget the captured "look and feel" to new models. The key contribution is a novel representation, namely the "style sheet", that is captured independently from a building's structure. It summarizes characteristic shape and texture patterns on the building. In the retargeting stage, a style sheet is used to decorate new buildings of potentially different structures. Consistent face groups are proposed to capture complex texture patterns from the example model and to preserve the patterns in the retarget models. We will demonstrate how to learn such style sheets from different Manhattan world buildings and the results of using them to generate novel models.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling packages

## 1. Introduction

Creating building models of a particular style is important to many applications such as digital entertainment and urban planning. Approaches such as procedural modeling [PM01] and example based model synthesis [MSK10] significantly reduce the cost of producing a large number of models of a common style. Inverse procedural modeling methods [MZWVG07, BWS10, ARB07, LCOZ*11] are also proposed to bridge the gap between intuitive user experience and fast model generation.

The focus of this paper is the generation of Manhattan World (MW) buildings. They are widely seen and are characterized by the predominance of three mutually orthogonal directions. Assuming Manhattan World

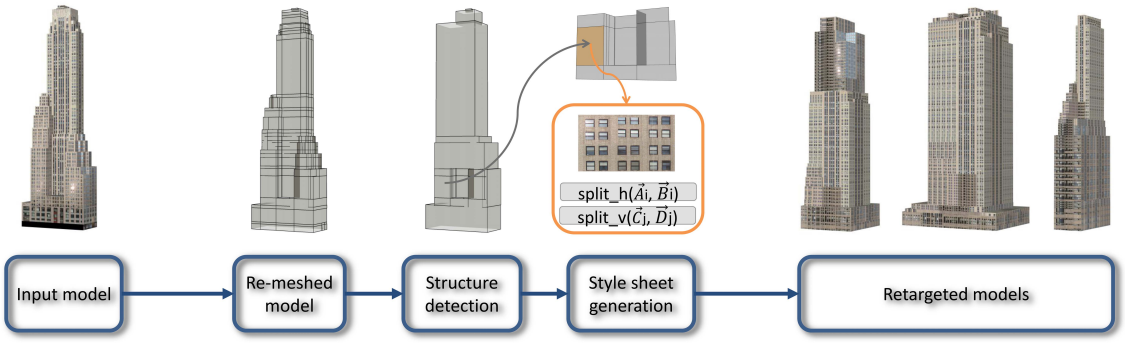



**Figure 3:** *Our system takes a textured 3D model as the input. In the capturing stage, our system remeshes the input model, detects its structure and generates a style sheet. In the retargeting stage, the style sheet is used to decorate novel models of different structures.*

**Figure 2:** *MW buildings usually contain characteristic decorations. Left: instead of having a flat facade, the building's shape evolves from the bottom to the top. Right: textures may also change over a building's surface. The facades here are decorated with different textures.*

is common practice in vision and graphics research [CY99, FJZ05, FCSS09, VAB10, CAAB12, BYMW13] and regularizes the problem solution. However, stylizing the appearance of these buildings is still challenging: First, unlike low-rise buildings, their appearance can vary greatly between floors. Figure 2 shows examples of shape and texture variations that would be difficult to describe by a unified procedural rule. Second, in the real world one MW building can be structurally very different to the next, graphics tools that simply scale a model do not permit a richness of variation for the generation of new models. Last but not the least, existing retargeting methods [ARB07, LCOZ*11] require manual labeling for the structure and for the detail patterns of the 3D input model.

We propose a method that captures and retargets the "look and feel" of Manhattan world buildings. The contributions are:

- A novel representation, namely the "style sheet", for describing the decoration (shape and texture details) of MW Buildings.
- A method for capturing the style sheet from an example building, with minimal user input.
- A method for automatically applying the style sheet across buildings of different structures.

Our method requires minimum manual input: a user only needs to align the example model to a standard view and separate the bottom and top floors from the main body of the building. Figure 3 gives an overview of our system. It has two main components: capture and the retargeting.

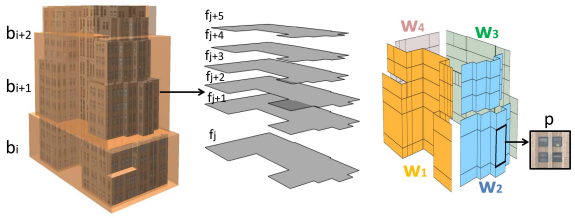
The capturing stage remeshes the input model, detects its structure and generates a style sheet. The remeshing process converts the input model into a sequence of floors that are linked by vertical faces. We then detect the structure of the building as a stack of blocks. Finally a style sheet is generated to capture the model's shape and texture details.

The retargeting stage generates new models of similar appearance but different in structure. The challenge we address is to apply the style sheet of a source building to the structure of a target building by automatically mapping the shape and texture details from the source blocks to the target blocks.

## 2. Related Work

The topic of generating multiple new building models in a common style is related to the following three fields: procedural modeling, example-based model synthesis, and inverse procedural modeling. Here we briefly review some of the representative works.

*Procedural modeling*. [PM01] propose a system that generates large-scale cities. They also noted that building structures usually do not reflect the growth process of a L-system. [WWSR03] invented split grammars to impose stricter spatial constraints for building generation. [MWH*06] generate detailed building shells by assembling basic shapes using operations such as scaling, translation,




(a) Blocks (b) Floors (c) Walls

**Figure 4:** *Some key terminologies used in this paper: (a)* ***b*** *for* ***blocks****, (b)* ***f*** *for* ***floors****, (c)* ***w*** *for* ***walls*** *and* ***p*** *for* ***faces****.*

rotation and split. Procedural modeling gives fast generation and is widely used in industry production [Cit08].

A common drawback of procedural modeling is its non-trivial process of rule specification. To achieve more intuitive user control, [TLL*11] use a high level specification, such as a sketch or a volumetric shape to guide procedural modeling. However, [TLL*11] is not designed for buildings and only generates models that consist of regular blocks.

*Example-based model synthesis*. Methods in this category use the idea of texture synthesis [WLKT09, LHL10] to preserve the local appearance of the given example in the generated model. For example, [MM11] stitch together pieces of model and incorporate the adjacency constraint in 3D to ensure that all of the pieces fit together seamlessly. [CLDD09] couple the manipulation of geometry and texture for reshaping and combining architectural models. Example-based approaches in general give a more friendly user experience, but suffer from the high computational cost of generating detailed models [Mer09].

*Inverse procedural modeling*. Methods in this category infer procedural rules from the example model. Regularities are used to capture building facades from images [MZWVG07, XFT*08, TKS*11, MWW12] or from 3D point clouds [PMW*08, LZS*11, SHFH11]. Local symmetry has also been explored to generate structurally sound 3D models [BWS10]. A related field to inverse procedural modeling is model abstraction. [MZL*09] simplify 3D geometric models using characteristic curves or contours. [NSX*11] use the Gestalt rules to summarize the structure of complex spatial arrangements found in architectural drawings.

One important application of inverse procedural modeling is to retarget the style of the example building to novel models. [ARB07] use a template grammar, extracted from a regularly structured model, to subdivide the floors of a new model. [LCOZ*11] decomposes a geometric model into groups of parts. Each group is labeled by a unique retarget attribute, such as scalable and replicable. Irregular structures of complex models can then be preserved by retargeting the groups in a sequential way. Recently [BYMW13] proposed an interactive system that allows user to generate and explore visually plausible building layouts.

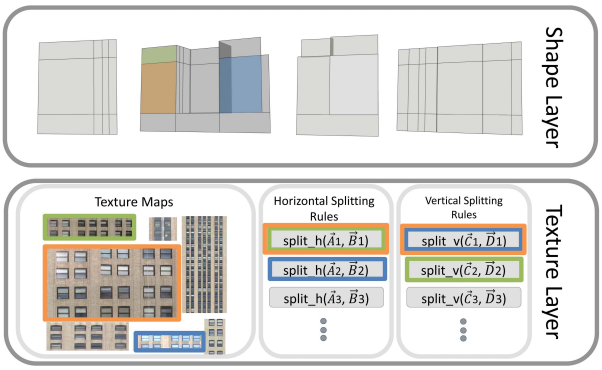



**Figure 5:** *A style sheet contains a shape layer and a texture layer. The shape layer stores shape details as walls. The texture layer contains texture maps and procedural splitting rules. Each face on a wall is assigned with a texture map and a pair of horizontal/vertical splitting rules (colour coded).*

In common with [ARB07, LCOZ*11], our objective is to transfer the style of a given building to new models. By limiting our system to MW buildings, we reduce the amount of manual input needed to capture the style of a model. In contrast to [LCOZ*11], we separate the structure of the model from its decoration details so the style can be transformed to models of different structures. Unlike [ARB07], we use a style sheet to capture multiple architectural patterns across a building model.

## 3. Terminology

In this section we first introduce the terminology (Figure 4) and then define the style sheet (Figure 5).

A model is vertically decomposed into a set of **blocks**. Figure 4-a shows three example blocks (in orange). Each block $b$ is a vertical extent containing several **floors** with similar floor area. Figure 4-b shows the floors contained by the middle block. Each floor $f$ is represented as a list of edges that are linked head-to-tail. Blocks are chosen such that there is only a small change in area between the floors within. The mesh contained by a block can also be decomposed into four vertical **walls** ($w_1$ to $w_4$ in Figure 4-c). Note that what we call a wall is not necessarily planar. For example $w_1$ in Figure 4-c is a typical non-planar wall: It has non-zero extent in all three principle dimensions (a height, a width and a thickness). Each wall $w$ is subdivided by the floors into horizontal strips of one or more **faces** ($p$ in Figure 4-c). Finally, each face $p$ carries its own 2D **texture map**.

Now we define our proposed style sheet using the above terminology. A style sheet has two layers: a shape layer and a texture layer (Figure 5). The shape layer captures the shape details of the building by storing the walls of each of its building blocks. In the retargeting stage these walls are mapped to target models using similarity transformations.

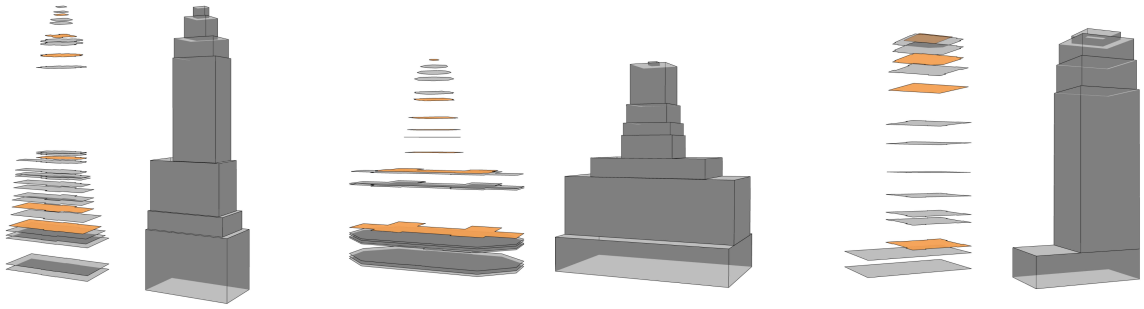


**Figure 6:** *We detect the structure of a building using its floor shapes. From all floors, we find the ones that bring significant changes to the building's cross sectional area (in orange colour). A base model is generated using building blocks that are separated by these significant floors.*

Every face on a wall is assigned a 2D texture map and a pair of splitting rules from the texture layer (indicated by the coloured boxes in Figure 5). The detection of the splitting rules is important because it allows procedurally generating windows for the retargeted model. Without these rules the generated windows are likely to have inconsistent shapes. In this paper we improve the detection for splitting rules using consistent face groups. For example, the orange face and the green face in Figure 5 are members of the same vertically consistent group, so they are assigned the same horizontal splitting rule; the orange face and the blue face are members of the same horizontally consistent group, so they are assigned the same vertical splitting rule.

## 4. Model Capture

Now we explain how to detect the structure of a model and capture its style sheet. The examples are highly user-rated models selected from the *Google 3D warehouse*. These models represent real and permanent building structures with textures of real photographs. Figure 3 shows an example model.

### 4.1. Remeshing

The input models need some pre-processing before they can be used for analysis. First, we choose the standard view of the input model by manually aligning the front of the building with the xy plane in the world space. For consistent parameter setting across models of different sizes, we also scale the model to unit height. Because these models are handcrafted, they sometimes have artefacts such as redundant or mis-aligned vertices. To address this, we remesh the model into a sequence of floors that are linked by vertical faces. Figure 6 demonstrates floors of three example models and Figure 3 has an example of a remeshed model.

To remesh the model, we first generate a set of horizontal planes (floors) from the vertices of the input model: we start from creating an adjacency matrix by connecting a pair of vertices if the difference between their heights is smaller than a threshold (set to 0.0005 of the building's height). The adjacency lists of the matrix are used to perform a vertical alignment: all vertices belonging to the same list are shifted to a horizontal plane at their average height. By intersecting this horizontal plane with the model, we acquire the edges that complete the shape of a floor. Redundant vertices and edges are eliminated using a minimum neighborhood distance check. Figure 6 shows the acquired floors of three example buildings.

We then use vertical faces to connect floors: each edge on a floor generates one vertical face by projecting itself to the floor below. Two floors are fully connected once all edges on the upper floor have been projected (Figure 4-c). Doing so will lose non-MW features such as non-vertical walls, peaked roofs or complex facade ornaments (Figure 16). However for MW buildings we find it generally gives good approximations.

Finally we add texture to the remeshed model. We project faces in the original model to the remeshed model with their textures. Each face is projected along its own normal direction. The texture on the remeshed model is then interpolated from the projections. In practice multiple textures might project onto the same face in the remeshed model. In this case we select the projection that covers the largest area of the face.

### 4.2. Structure Detection

To detect the structure of the model, we are interested in the floors that result in significant changes to the building's shape. These floors (the orange ones in Figure 6) "cut" the building into a few blocks, where floors significantly change their shapes across two blocks but remain relatively consistent inside each one.

Let us name the sequence of floors in a remeshed model $\{f_1, ..., f_m\}$. To find the cuts, we calculate the ratio between the areas of two successive floors by dividing the larger floor by the smaller floor: $r_i = \frac{\max(f_i, f_{i-1})}{\min(f_i, f_{i-1})}$. A naive solution is to keep floors that have $r$ larger than a certain threshold.

But this would require the threshold to be set manually for each building as the degree of floor variation varies from one building to another. Instead we use local maxima to identify the cuts. This means a cut should satisfy $r_i > r_{i-1}$ and $r_i > r_{i+1}$. Three buildings are shown in Figure 6 where the cuts are rendered in orange and the non-cut floors are rendered in gray. Blocks are generated by vertically extruding the bounding boxes of the cuts. All blocks together form a base model of the building that summarizes its structure.

### 4.3. Style Sheet Capture

Now we explain how to acquire a style sheet. We first introduce the basic idea of having a shape layer and a texture layer in the design of a style sheet, then explain how to use consistent face group to improve the detection of procedural rules for complex facades.

#### 4.3.1. Shape Layer

The shape layer stores the shape details of the building. We break the mesh contained by each block into four walls: the front, the right, the back and the left ($w_1$ to $w_4$ in Figure 4-c). The advantage of having these walls is their thicknesses are adjustable in the later retargeting stage (Figure 11).

In order to acquire the walls, we break each floor into four pieces – that is one piece for each side of the block. A wall contains all floor pieces from the same side. In practice we use the four corners of a floor's bounding box and identify each corner a nearest neighbor vertex on the floor contour. These four vertices are used as the break points.

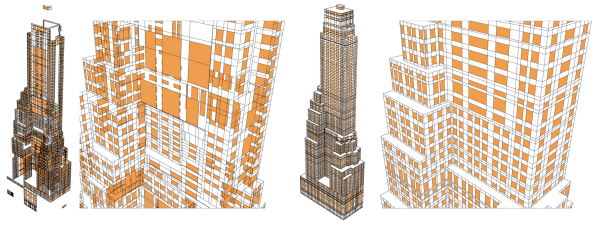
#### 4.3.2. Texture Layer

A texture layer stores texture maps and procedural rules. Each face on a wall is associated with one texture map and a pair of splitting rules, one horizontal and one vertical (Figure 5, bottom). These rules split the texture map into two types of element: the insertable (windows) and the scalable (non-window part of the facades). In this way new textures can be procedurally generated: the insertable elements can be added or removed depending on the size of each retargeted face and the scalable elements fill the space between them.

#### 4.3.3. Consistent Face Groups

A MW building usually contains different window patterns that generate the rhythm of its facade (Figure 2-b). Consequently it is difficult to use a single pair of procedural rules to model the texture of the entire building. It is not optimal to detect procedural rules from individual faces as inconsistent texture layouts are likely to be generated in the retargeting stage (Figure 7-a).

Our solution is to identify groups of faces that have consistent shape and texture properties, such that there is one splitting rule for each group. Using this group-based rule detection, we are able to capture complex decorations from the example building and generate tidy facade layouts for the new models (Figure 7-b).



(a) (b)

**Figure 7:** *Orange patches indicate insertable elements. (a) The layout generated using splitting rules detected from each face. Note the layout is not only badly aligned, but also missing from the faces that do not have rules detected. In contrast, group based rule detection (b) is able to generate a well-aligned layout.*

We first identify horizontally consistent groups. We note all faces between two successive floors have the same height, which is the vertical gap between the two floors. Moreover, windows on these faces are usually aligned so a single vertical splitting rule is applicable. Hence a horizontally consistent group can be simply defined as all faces between two successive floors. Figure 8-b shows horizontally consistent groups using a colour-coded facade, where each colour represents one group.

Vertical consistency also exists, for example, a stack of faces that share the same width and window pattern (Figure 8-c). In order to identify vertically consistent groups, we link faces by following a bottom to top traversal: for each face $p_i$ we find its neighboring face $p_j$ from the floor above, and link $p_i$ and $p_j$ if they have similar shape (width) and texture. The shape similarity is calculated as

$$\phi_{i,j}^{shape} = \frac{\cap(\zeta_i, \zeta_j)}{min(\zeta_i, \zeta_j)} \tag{1}$$

where $\zeta$ is the width of a face and $\cap(\zeta_i, \zeta_j)$ is the width of the two faces' overlapping section. The texture similarity is calculated as the correlation between two colour signals:

$$\phi_{i,j}^{texture} = corr(\omega_i, \omega_j) \tag{2}$$

where each signal $\omega$ is calculated by averaging the pixel columns in the texture map. A link is added if $\phi_{i,j}^{shape} > 0.9$ and $\phi_{i,j}^{texture} > 0.5$.

#### 4.3.4. Splitting Rules

Now we detect splitting rules based on consistent face groups. Specifically, we acquire one vertical splitting rule

(a) (b) (c)

**Figure 8:** *(a) An example model. (b) Horizontally consistent groups. (c) Vertically consistent groups.*

from each horizontally consistent group, and one horizontal splitting rule from each vertically consistent group. In practice we average texture maps of all group members for the detection of the rules. In this way the acquired rules generalize better for all group members. Figure 9 shows an example of an averaged texture map that is used to detect a horizontal splitting rule (vertical splitting rules can be detected in the similar way).

Our key assumption is that scalable elements (defined in Section 4.3.2, which are non-window part of the facades) should have a homogenous colour that is different from the insertable elements (such as windows), hence large colour variance should be detected for image columns that intersect with both types of elements.

To detect the rule, we first calculate a discrete horizontal colour variance signal $var(i)$ (Figure 9, top) from image columns of the texture map (Figure 9, middle). The goal is to classify the signal into a sequence of insertable sections and scalable sections (the blue and the orange labels in Figure 9, bottom). This is essentially a binary 1D labeling problem. Let $l(i) = 1$ mean the label of an insertable element and $l(i) = 0$ mean the label of a scalable element, we use dynamic programming to find the optimal labeling by minimizing the following energy function:

$$l = argmin(\sum_{i=1}^{n} E_d(l(i)) + \lambda \sum_{i=2}^{n} E_s(l(i-1), l(i))) \qquad (3)$$

Here $i$ is the index for each place in the signal, $n$ is the length of the signal and $\lambda = 0.025$ is the relative weight of two terms: the data term $E_d$ and the smoothness term $E_s$.

The data term is the cumulative distance of each value to its assigned cluster center. We use k-means (k = 2) to initialize the two cluster centres $\mu_0$ and $\mu_1$ from the signal. The data term is then calculated as:

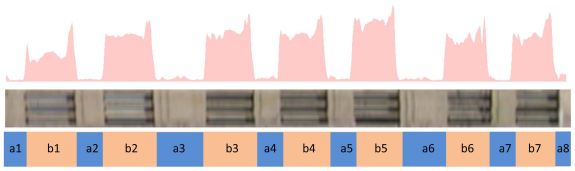



**Figure 9:** *The colour variance signal (top) is used to detect a horizontal splitting rule. The colour band (bottom) indicated the detected rule, which successfully splits the texture map (middle) into insertable (orange) and the scalable (blue) elements.*

$$E_d(l(i)) = \begin{cases} |var(i) - \mu_0| & \text{if } l(i) = 0 \\ |var(i) - \mu_1| & \text{if } l(i) = 1 \end{cases} \qquad (4)$$

The smoothness term $E_s$ promote spatial coherence for the labels of adjacent signal values:

$$E_s(l(i-1), l(i)) = \begin{cases} 1 & \text{if } l(i-1) \neq l(i) \\ 0 & \text{if } l(i-1) = l(i) \end{cases} \qquad (5)$$

As in [MWW12], we minimize Equation 3 by accumulating the energy over the stages followed by a reverse backtracking of a minimum energy path. Figure 9 shows the result labels: blue means scalable and orange means insertable. Notice the colour bands correlate well with the texture map.

Dynamic programming can sometimes produce fragmented labels. In this case a user-specified objective window size can be used to merge insertable sections whose lengths are significantly smaller than the objective window size.

We write the horizontal splitting function as $[S_n, S_r] = split\_h(\overrightarrow{A}, \overrightarrow{B}, W)$. The input $\overrightarrow{A} = [a_1, a_2, ...a_{m+1}]$ and $\overrightarrow{B} = [b_1, b_2, ..., b_m]$ are vectors encoding the splitting rule. They essentially describe the detected wall and window widths from the input texture (see Figure 9). $W$ is the width of the new face to be split. The output is a pair of numbers $[S_n, S_r]$, where $S_n$ is the number of insertable elements and $S_r$ is the scaling factor for the scalable elements. We will explain the splitting function in more detail in Section 5.2. For highly regular textures we could summarize vectors $\overrightarrow{A}$ and $\overrightarrow{B}$ using their average values. However in practice we find it is useful to keep the full vectors so irregularities can be retained.

Similarly, a vertical splitting rule can be detected using image rows of the texture maps. Its corresponding splitting function is written as $[S_n, S_r] = split\_v(\overrightarrow{C}, \overrightarrow{D}, H)$, where $\overrightarrow{C}$ and $\overrightarrow{D}$ encodes the vertical splitting rule, and $H$ is the height of the face to be split.

We can use the splitting functions to re-generate the texture layout of the input model (Figure 7-b). Compared

to the result of using rules detected from individual faces (Figure 7-a), consistent face groups produce significantly better aligned texture layout.

## 5. Style Retargeting

Our purpose in acquiring the style of a building is to transfer it onto new models. Separating the structure from the style sheet allows retargeting between models of different structures, for example Figure 10-a and Figure 10-b. Such base models can be created manually or by applying our structure detection algorithm (4.2) to existing building models.

Our retargeting contains two steps: shape retargeting and texture retargeting. Next we explain them in detail.

### 5.1. Shape Retargeting

For shape retargeting, our method selects a good match from the source model for each block in the target model. It then maps the shape contained in the selected source block onto its matched target block.

#### 5.1.1. Block Matching

We use a greedy matching algorithm that follows a bottom to top traversal. For a source block $i$ and a target block $j$ we calculate a shape cost $\Psi_{shape}(i,j)$ and a volume cost $\Psi_{volume}(i,j)$. We seek to minimize the transformation cost defined as:

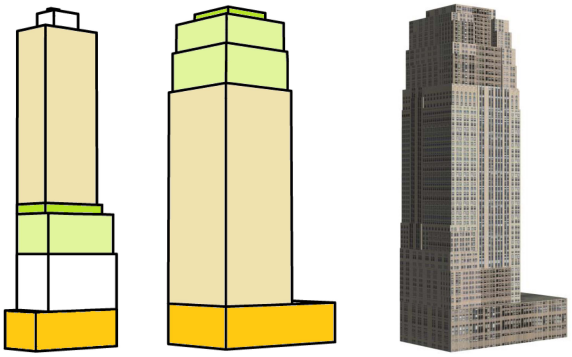
$$\Psi(i,j) = \Psi_{shape}(i,j) \times \Psi_{volume}(i,j). \tag{6}$$

The shape cost $\Psi_{shape}(i,j)$ penalizes potential matches where one or more of the dimensions differ greatly between source and target blocks. Let $[\eta_1,\eta_2,\eta_3]$ denote the width, height and depth of a building block. We first normalize the sum of each block's three dimensions to a unit one: $\bar{\eta_1} = \frac{\eta_1}{\eta_1+\eta_2+\eta_3}$, $\bar{\eta_2} = \frac{\eta_2}{\eta_1+\eta_2+\eta_3}$ and $\bar{\eta_3} = \frac{\eta_3}{\eta_1+\eta_2+\eta_3}$. Then we calculate $\Psi_{shape}(i,j)$ as

$$\Psi_{shape}(i,j) = \max\left(\frac{\bar{\eta_{1i}}}{\bar{\eta_{1j}}}, \frac{\bar{\eta_{1j}}}{\bar{\eta_{1i}}}\right) \max\left(\frac{\bar{\eta_{2i}}}{\bar{\eta_{2j}}}, \frac{\bar{\eta_{2j}}}{\bar{\eta_{2i}}}\right) \max\left(\frac{\bar{\eta_{3i}}}{\bar{\eta_{3j}}}, \frac{\bar{\eta_{3j}}}{\bar{\eta_{3i}}}\right) \tag{7}$$

The volume cost penalizes matches between blocks of different relative volumes. The relative volume of a block is as: $vol_i = \frac{\eta_{1i}\times\eta_{2i}\times\eta_{3i}}{\sum_{k=1}^{n}\eta_{1k}\times\eta_{2k}\times\eta_{3k}}$. The volume cost is then calculated as $\Psi_{volume}(i,j) = \frac{max(vol_i,vol_j)}{min(vol_i,vol_j)}$.

The final cost $\Psi(i,j)$ is the product of the shape term and the volume term. For each target block, our greedy solution selects the source block that results in the lowest cost. Figure 10 shows the matching result between two buildings, where the matched blocks are labeled by the same colour. Note the matching does not have to be one to one.



(a) (b) (c)

**Figure 10:** *Our system automatically retargets the appearance of a building to a new one. For each building block in the target model (b), one block from the source model (a) is selected based on minimizing a transformation cost. The matches are labelled in the same colour. Our style sheet then generates decoration details on the target model. The final textured model (c) is shown on the right.*

#### 5.1.2. Shape Mapping

For each matched pair, we map the shape contained in the source block to the target. Specifically, we parameterize a similarity transformation using block corners as the correspondences. This transformation is used to map all walls from the source block to the target.

However, in practice artefacts can be raised by incompatible building blocks. Figure 11-a shows an example, where the orange wall in the upper block reaches over the blue wall in the lower block. This contradicts our experience that the upper part of a MW building is usually supported by its lower part.

To remove such artefacts we change the inner boundary of the blue wall (Figure 11-b). Let $\delta$ denote the thickness of a wall, that is the distance between its inner and outer boundaries. We simply scale the thickness of the blue wall to $\delta'$, which is a portion (set to 0.9 in this paper) of the distance between the orange and the blue walls' outer boundaries.

In practice we find it is not helpful to retarget the top or the bottom of a source building to the main body of a target model. The reason is that these parts of a building usually have characteristic shapes and textures that do not exist elsewhere. Hence we force the bottom of the source model to be retargeted to the bottom of the target model, and the top of the source model to be retargeted to the top of the target model. In practice, users need to manually label the bottom and top blocks so they can be separated from the building's main body. Note this labeling task and the standard view selection in the preprocessing step (Section 4) are the only required manual inputs of our system.

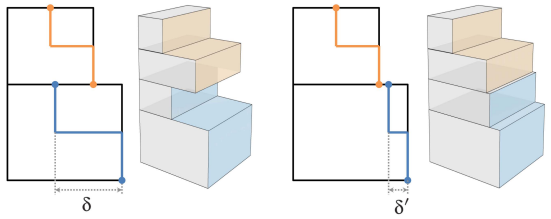



**Figure 11:** *(a) The orange wall reaches over of the blue wall. Here we show the problem from a 2D side view as well as a 3D view. (b) We reduce the thickness of the blue wall to remove such a "overhanging" artefact.*

### 5.2. Texture Retargeting

The texture retargeting process generates texture for each face using splitting rules. To preserve consistent texture layout, the same horizontal (or vertical) splitting rule is applied on all the faces in the same vertically (or horizontally) consistent group. Each face uses its own texture map to generate insertable/scalable element patches, so the richness of the model's appearance is preserved.

However, it is possible to have inconsistent texture layout across different blocks because blocks are handled independently of each other. Figure 12-a shows an example where textures are misaligned. We solve this problem using two adjustment processes: face snapping and texture re-assignment.

#### 5.2.1. Face Snapping

The face snapping process aligns faces in the vertical direction. Following a bottom to top traversal, for each face $p_i$ we snap its vertical edges toward its neighboring face $p_j$ from the floor below. Figure 12-b shows an example of a snapped model.

We snap faces sequentially for each of the four sides of the building (front, right, back, and left). The idea is to shift vertices on the upper floor towards their neighbouring vertices on the lower floor, so the associated faces are aligned. In practice the orthogonal projection of each side of the building is used as the reference view (see Figure 12) for snapping, so a vertex only needs to be shifted towards left or right to align with its neighbouring vertex on the lower floor.

In practice the snapping process is implemented using dynamic time warping (DTW). DTW finds an optimal match between two sequences with certain restrictions, and warps one of the sequences non-linearly towards the other. In our case the two (discrete) sequences are the lists of vertices from the two adjacent floors. The distance between the vertices is calculated as the Euclidean distance of their horizontal locations from the reference view. We also add a locality constraint to prevent the vertices from being over-shifted, that is, we require that if vertex $v_i$ is matched with vertex $v_j$, then $|v_i - v_j|$ is no larger than a window parameter $\kappa$. In this paper we set $\kappa$ as 0.2 of the width of the building from the reference view. Note in practice some faces may be squeezed to zero width and if so they will be removed from the model.

#### 5.2.2. Texture Re-assignment

Although the snapped faces are geometrically aligned, they may have inconsistent textures as indicated by the fragments in the colour-coded facade (Figure 12-b). We use a texture re-assignment process to solve this problem. Specifically, the texture maps of large consistent groups are assigned to the smaller groups. The result is a fragment-free colour-coded facade (Figure 12-c).

#### 5.2.3. Procedural Generation

Finally we split each face using the splitting functions. Suppose we need to horizontally split a face using $[S_n, S_r] = split\_h(\overrightarrow{A}, \overrightarrow{B}, W)$. Recall $A = [a_1, a_2, ...a_{m+1}]$ are the widths of scalable elements and $B = [b_1, b_2, ...b_m]$ are the widths of insertable elements, and $W$ is the width of the face to be split. To split we first estimate $S_n$, which is the number of insertable elements (windows) that fits $W$. We then calculate $S_r$, which is the scaling factor for filling the remaining space with the scalable elements.

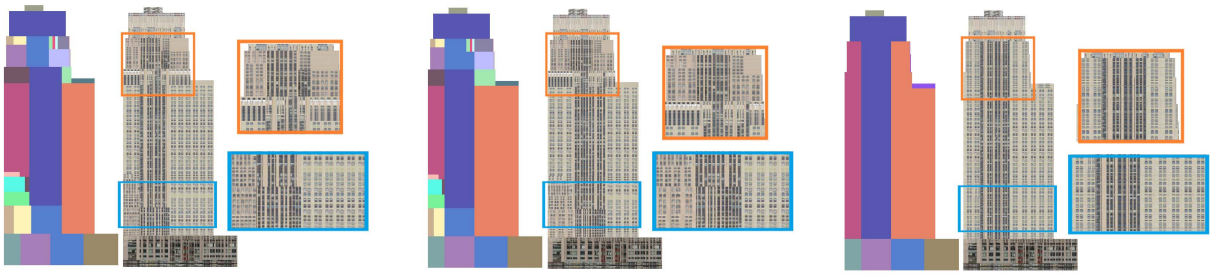
To calculate $S_n$, we first calculate $x$, the maximum repeats for the original texture map and $y$, the maximum number of windows that can be inserted into the remaining space. This can be calculated by maximizing the following function with $W$ as the upper bound:

$$(x, y) = \arg\max x(\sum_{i=1}^{m} b_i + \sum_{i=1}^{m+1} a_i) + \sum_{j=1}^{y} b_j \qquad (8)$$

Here $x$ can be calculated as the quotient of $\frac{W}{\sum_{i=1}^{m} b_i + \sum_{i=1}^{m+1} a_i}$, and $y$ is the maximum number of insertable elements that can fit into the space $W - x(\sum_{i=1}^{m} b_i + \sum_{i=1}^{m+1} a_i)$. The total number of windows is then calculated as $S_n = x \times m + y$, which means in the generated texture, $[b_1, b_2, ..., b_m]$ will repeat $x$ times and $[b_1, b_2, ...b_y]$ will be added to the tail.

The remaining space, $W - x(\sum_{i=1}^{m} b_i + \sum_{i=1}^{m+1} a_i) - \sum_{j=1}^{y} b_j$, is to be filled by the scalable elements. As each insertable element is adjacent to two scalable elements, the total number of scalable elements is $x \times (m+1) + y + 1$. This means in the generated texture, $[a_1, a_2, ..., a_{m+1}]$ will be repeated $x$ times and $[a_1, a_2, ..., a_{y+1}]$ will be added to the tail. Hence the scaling factor is

$$S_r = \frac{W - x(\sum_{i=1}^{m} b_i + \sum_{i=1}^{m+1} a_i) - \sum_{j=1}^{y} b_j}{x \sum_{i=1}^{m+1} a_i + \sum_{j=1}^{y+1} a_j} \qquad (9)$$



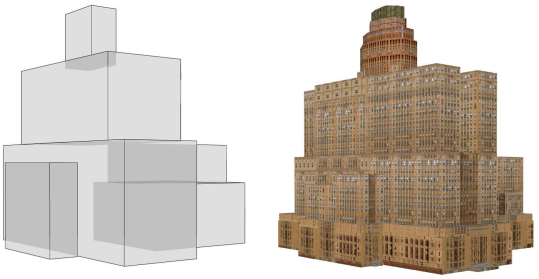
(a) (b) (c)

**Figure 12:** *(a). The retargeted model may have mis-aligned faces across adjacent blocks. To address this problem we first snap faces along the vertical direction (b) and then re-assign texture to enforce texture consistency (c).*

## 6. Results and Discussion

This section demonstrates our results and discusses the advantages of our method compared to related approaches such as [ARB07] and [LCOZ*11]. Figure 14 shows our method is able to work with different types of Manhattan World buildings. Each row starts from an example model which is followed by five new models stylized using the example's style sheet. The last picture of each row is a close-up inspection of the top of the fifth model. One important advantage of our system is its ability to generate models with various floor shapes and textures. Specifically, we allow each horizontally consistent group to have its own vertical splitting pattern, and each vertically consistent group to have its own horizontal splitting pattern. This is be difficult to achieve using the regular pattern detected from one floor or one flat facade. It also requires non-trivial manual input for existing methods like [ARB07]. Note a similar idea has appeared in [MWW12], where the authors use so-called "coherence-groups" to improve window detection for a complex facade texture. The main difference is [MWW12] is an interactive solution, and our method is automatic – given the facade texture is segmented by the mesh and the window layout in each segment is grid-like. Note although this paper uses a colour variance based method for local texture rule detection, other repetition detectors can be used when the colour variance assumption does not work well.

Separating the structure of a model from its appearance details is another advantage of our system. Our system uses building blocks to match the structures of different models and automatically transfers the shape and texture details from the source to the targets. Compared to the "retargeting by scaling" approaches such as [LCOZ*11], our method works better with models of different structures. In this paper, the base models are acquired using the structure detection method introduced in section 4.2. In practice, our system also works with manually made or procedurally generated base models, as long as they are represented by building blocks. Figure 13 shows an stylized building generated from a manually created base model.



(a) (b)

**Figure 13:** *An example of stylizing a base model that has building blocks connected in both vertical and horizontal directions. The base model (a) is manually made and the style sheet is the same as "style two" in figure 1.*

Our system also allows users to edit the output models. For example, a user can interactively change the shape of the building by resizing its building blocks. Figure 15 shows an example of altering the size of an existing model. Notice our system generates textures that fit the resized model using procedural rules. This avoids artefacts such as distorted windows from using image scaling, or mis-aligned windows from using traditional example based texture synthesis.

The major free parameters of our system are fixed (values are given throughout the paper) for different buildings. In practice our system spends most of its running time to acquire the style sheet – on average about 30 seconds for a textured model with 5000 polygons using an Intel i7 processor. It takes about 10 seconds to generate a new model of the same complexity.

In this paper the structure of a building is detected by the significant "cuts" along the vertical direction. Indeed it is possible to cut a model along other directions, especially along the other two dominant directions of the Manhattan World buildings. We choose to only use horizontal cuts for two reasons: First, the shape of a MW building mainly varies with height so it is easier to identify significant cuts from the floors. Second, the floors have interesting properties. For example, the windows between two successive floors

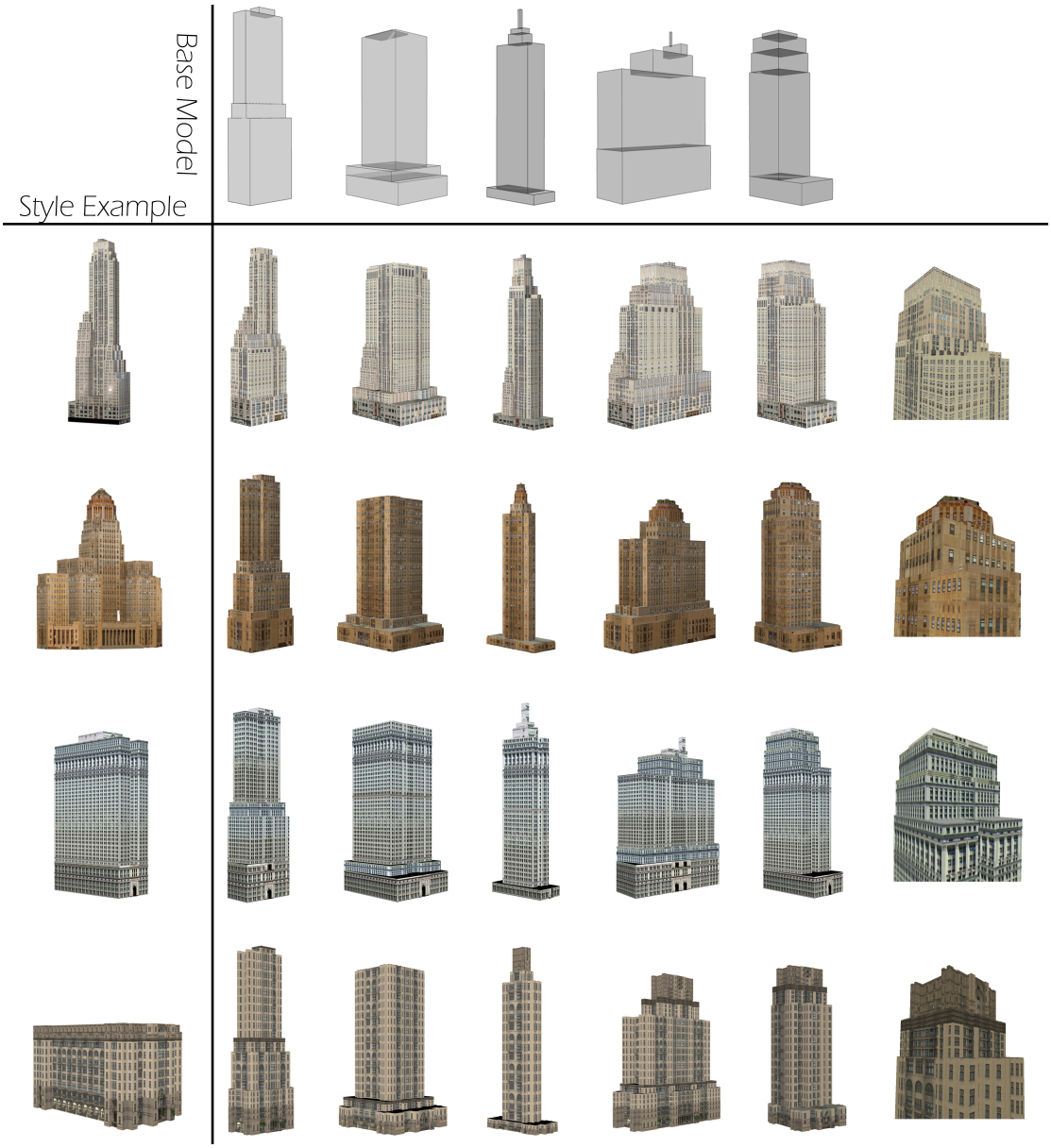



**Figure 14:** *A set of stylized Manhattan world buildings generated using our system.*

are usually levelled, so horizontal consistent groups are naturally produced.

## 7. Limitations

Finally we discuss the range of buildings that our system is designed to work with. Choosing Manhattan World regularizes the problem solution. In particular it makes the automatic acquisition and retargeting of a building's appearance style a feasible task. In consequence our current method will have difficulties with models which have non-MW world features, such as non-vertical walls and peaked roofs. Figure 16 shows an example of such a building. Our remeshing process may lose complex facade details due to our use of vertical faces. An interesting future work is to include non-MW world features as terminal shapes in our system.

Another limitation comes from the building blocks that are used for retargeting. Building blocks can not accurately describe L-shapes, T-shapes, or polygons that are not four-sided, such as the hexagon example in Figure 16. A possible solution is to include different types of building blocks for abstracting these building models. In this paper

we only use horizontal planes to cut the model. This works fine for MW buidings whose shape mainly varies along with height. A multiple directional structure detection algorithm may be required for working with buildings of more complex shapes. The algorithm also requires that the texture can be split into a grid, which is usually applicable to window textures. If the building contains some special structure, e.g. important ornaments, or the windows are not placed in a grid style, then user assistance is needed to improve the acquisition of procedural rules. Currently our system acquires splitting rules from texture images. This works fine for models of medium level shape complexity where such details are stored as textures. For models that have complex shape details, we would need to use a 3D repetition detector or even interactive methods as [LCOZ*11] did.

The vertical consistency of retargeted textures is improved using face snapping and texture re-assignment. However, the improvement comes with a cost of texture richness (due to the texture re-assignment operation). This can be identified by comparing the colour coded facades in Figure 12-b and Figure 12-c. In practice users can balance the consistency and the richness of the generated texture by manually setting a threshold for the maximum size of the fragments to be re-assigned with new textures. In the future it is interesting to explore high-level semantics for a better automatic solution.

Last but not the least we discuss issues related to user interaction. We first discuss situations that user interactions are required, and then discuss situations that user interactions can be used to improve the results.

In this paper the required user interactions are 1) choosing the standard view of the model in the re-meshing process, 2) separating the top and bottom from the rest of the building. While the first can be replaced by a PCA analysis, the second is crucially important to achieve the results demonstrated in the paper. For example, the base of the first example building in Figure 14 has doors and shops. It would be implausible to retarget them to the main body of the generated models. The second example building has a stylish top. It should be retargeted to the top of other buildings (see the third, fourth and fifth generated buildings in the same row). In practice user interaction is needed to ensure such features are correctly identified in the structure detection stage and properly matched in the retarget stage.

Next we discuss several cases that user interaction can be adapted to remove artefacts in the automatic solution. In this paper the quality of the input model has a strong impact on the result: a modeler can subdivide a face into an arbitrary number of faces, and carelessly crafted model will contain mis-aligned faces that causes false grouping and inconsistent retargeting. Currently a user need to remove such artefact by manually editing the input model. In the future we will investigate better solutions to this issue, for example, by automatically splitting and merging the faces in the remeshed model.

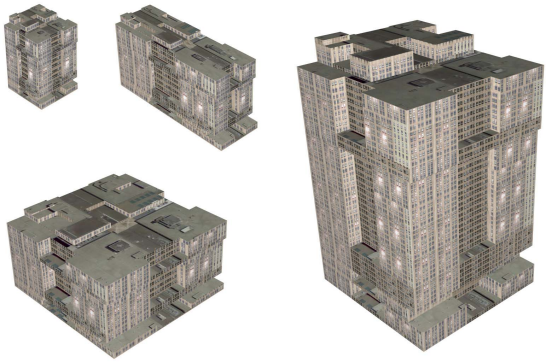
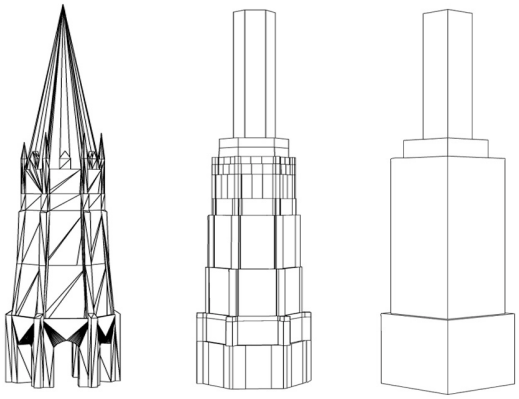


**Figure 15:** *A user can change the shape of the building by operating on its blocks. Here we show examples of resizing a building. The decoration layout of the resized model is well aligned using our style sheet.*



(a) (b) (c)

**Figure 16:** *One limitation of our system is that it does not work well with buildings which have non-MW world features. (a) shows a model of the Basilique Saint-Michel de Bordeaux. Some of its features, such as the non-vertical edges and hat roofs are missing from the remeshed model (b). Meantime, its hexagonal shape body is not suitable for being abstracted using four-sided building blocks (c).*

We use automatic face snapping to optimize the appearance of the retarget buildings. However, unsnapped fragments, as on the lower left part of the building in Figure 12-b, can still occur. Although we address this problem by automatically re-assigning the texture to the fragments, user interaction could be an alternative solution. For example, let users have the option to “paint” on the colour coded facade to personalize the texture layout.

## 8. Conclusion

We propose the style sheet as a novel representation for Manhattan World buildings, and use it for model stylization. Our method is able to handle multiple architectural patterns across a building, and retarget the captured style to models of potentially different structures. Our users will also benefit

from reduced manual input for inverse modeling, with simultaneous capturing of shapes and textures.

We have demonstrated the use of our system on different types of MW buildings, and the generation of new models of the same style. Our output models are as visually plausible as the input examples which are suitable for 3D map applications such as the Google Earth. However, the quality of the output model cannot surpass the input models. In the future we would like to use high resolution facade images, or even dense 3D point clouds to improve the visual quality for applications that requires close-ups.

Although the structure detection and style sheet generation techniques described in this paper rely on the MW assumption, the concept of base structure and style separation can be generalized to work with buildings of non-MW features as future work. It would also be interesting to make our method scalable to different shape complexities by fusing 2D and 3D repetition detections.

## 9. Acknowledgements

We thank reviewers for their valuable suggestions. We thank Darren Cosker, Peter Hall, David Pickup and Thomas Saunders for inspiring discussions. We also thank University of Bath for supporting this work.

## References

[ARB07] ALIAGA D. G., ROSEN P. A., BEKINS D. R.: Style grammars for interactive visualization of architecture. *IEEE TVCG 13* (2007), 786–797. 1, 2, 3, 9

[BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *SIGGRAPH* (2010), 1–10. 1, 3

[BYMW13] BAO F., YAN D.-M., MITRA N. J., WONKA P.: Generating and exploring good building layouts. *SIGGRAPH* (2013), 1–10. 2, 3

[CAAB12] CARLOS A V., ALIAGA D. G., BENES B.: Automatic extraction of manhattan-world building masses from 3d laser range scans. *IEEE TVCG* (2012), 1627–1637. 2

[Cit08] Esri cityengine, http://www.esri.com/software/cityengine. 3

[CLDD09] CABRAL M., LEFEBVRE S., DACHSBACHER C., DRETTAKIS G.: Structure preserving reshape for textured architectural scenes. *Computer Graphic Forum* (2009), 469–480. 3

[CY99] COUGHLAN J. M., YUILLE A. L.: Manhattan world: Compass direction from a single image by bayesian inference. *ICCV* (1999), 941–947. 2

[FCSS09] FURUKAWA Y., CURLESS B., SEITZ S. M., SZELISKI R.: Manhattan-world stereo. *CVPR* (2009), 1422–1429. 2

[FJZ05] FRÜH C., JAIN S., ZAKHOR A.: Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images. *International Journal of Computer Vision 61*, 2 (2005), 159–184. 2

[LCOZ*11] LIN J., COHEN-OR D., ZHANG H. R., LIANG C., SHARF A., DEUSSEN O., CHEN B.: Structure-preserving retargeting of irregular 3d architecture. *SIGGRAPH Asia* (2011), 183:1–183:10. 1, 2, 3, 9, 11

[LHL10] LEFEBVRE S., HORNUS S., LASRAM A.: By-example synthesis of architectural textures. *SIGGRAPH* (2010), 1–8. 3

[LZS*11] LI Y., ZHENG Q., SHARF A., COHEN-OR D., CHEN B., MITRA N. J.: 2d-3d fusion for layer decomposition of urban facades. *IEEE International Conference on Computer Vision* (2011), 882–889. 3

[Mer09] MERRELL P. C.: Model synthesis. *PhD Thesis, Stanford University* (2009). 3

[MM11] MERRELL P., MANOCHA D.: Model synthesis: A general procedural modeling algorithm. *IEEE TVCG 17*, 6 (2011), 715–728. 3

[MSK10] MERRELL P., SCHKUFZA E., KOLTUN V.: Computer-generated residential building layouts. *SIGGRAPH Asia* (2010), 1–12. 1

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *SIGGRAPH* (2006), 614–623. 2

[MWW12] MUSIALSKI P., WIMMER M., WONKA P.: Interactive coherence-based façade modeling. *EUROGRAPHICS* (2012), 661–670. 3, 6, 9

[MZL*09] MEHRA R., ZHOU Q., LONG J., SHEFFER A., GOOCH A., MITRA N. J.: Abstraction of man-made shapes. *SIGGRAPH Asia* (2009), 1–10. 3

[MZWVG07] MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based procedural modeling of facades. *SIGGRAPH 26* (2007), 85–94. 1, 3

[NSX*11] NAN L., SHARF A., XIE K., WONG T.-T., DEUSSEN O., COHEN-OR D., CHEN B.: Conjoining gestalt rules for abstraction of architectural drawings. 1–10. 3

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. *SIGGRAPH* (2001), 301–308. 1, 2

[PMW*08] PAULY M., MITRA N. J., WALLNER J., POTTMANN H., GUIBAS L. J.: Discovering structural regularity in 3d geometry. *SIGGRAPH* (2008), 1–11. 3

[SHFH11] SHEN C.-H., HUANG S.-S., FU H., HU S.-M.: Adaptive partitioning of urban facades. *SIGGRAPH ASIA* (2011), 1–9. 3

[TKS*11] TEBOUL O., KOKKINOS I., SIMON L., KOUTSOURAKIS P., PARAGIOS N.: Shape grammar parsing via reinforcement learning. In *CVPR* (2011), pp. 2273–2280. 3

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *Transactions on Graphics 30* (2011), 11:1–11:14. 3

[VAB10] VANEGAS C. A., ALIAGA D. G., BENES B.: Building reconstruction using manhattan-world grammars. In *CVPR* (2010), pp. 358–365. 2

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the Art in Example-based Texture Synthesis. In *Eurographics 2009, State of the Art Report* (2009), Eurographics Association, pp. 93–117. 3

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *SIGGRAPH* (2003), 669–677. 2

[XFT*08] XIAO J., FANG T., TAN P., ZHAO P., OFEK E., QUAN L.: Image-based facade modeling. *SIGGRAPH* (2008), 1–10. 3